Java Skill Exercise

Important Criteria

- Does the Java code provide an accurate and complete solution for the stated requirements?
- Are sound object-oriented principles used in the Java code?
- Are exceptions used to manage errors that can occur in the code?
- Do the JUnit tests provide adequate coverage of the Banking Portal functionality?

Scenario

You have been assigned to a development team that is building software that will be used to manage the accounts of a boutique bank client. The banking portal will limit the number of accounts managed. The portal must track information about all accounts so that it can report overall balance and how many new accounts can be added. The banking portal must provide publicly available methods to allow accounts to be added and removed from it (in which it updates its set of accounts). It must provide publicly available methods that report the maximum number of accounts under management, the remaining number of accounts that can added, and the total balance of all managed accounts. It must also provide a method to look up an account based on its account number.

A banking portal has a unique client ID. You must validate a client ID before assigning it to a new portal. You can validate a client ID using the provided `com.fidelity.utils.ClientIdValidator` class.

An attempt to add an account to a banking portal that has reached its limit of accounts should generate an exception to indicate an error condition. Also, an attempt to remove an account that is not in portal should generate an exception. An attempt to create an impossible banking portal or a portal with an invalid client ID should also generate exceptions.

There are three distinct types of accounts: debit, savings and GIC. Every account has an account number and a balance. A debit account has a fee which applies when its account balance is accessed. A savings account accrues 3% interest (disregard compound interest for this exercise). A GIC has a term period (in months), a fee, and 5% interest that applies only at the end of the term.

Task

Define the Java classes that are required to implement the functionality that is described above. Be sure to use sound object-oriented principles in your Java code. Create JUnit tests to adequately test all the classes you create. Whenever possible, you should write at least one positive and one negative test for each non-trivial public method that you write.

Note: Test cases can generate a unique client ID by calling `UUID.randomUUID().toString()`.